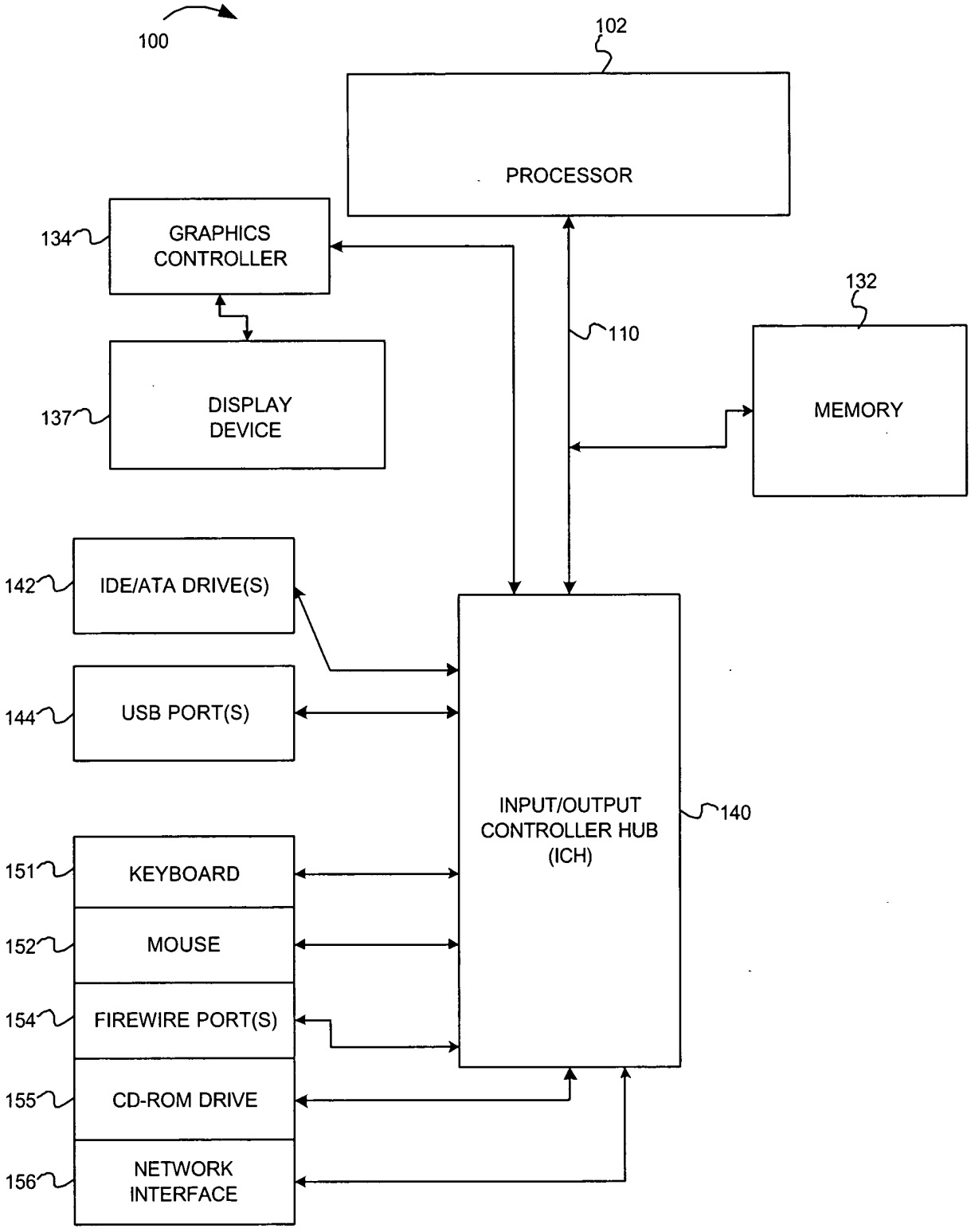
100
102
PROCESSOR
134
GRAPHICS CONTROLLER
132
110
137
DISPLAY DEVICE
MEMORY
142
IDE/ATA DRIVE(S)
144
USB PORT(S)
INPUT/OUTPUT CONTROLLER HUB (ICH)
140
151
KEYBOARD
152
MOUSE
154
FIREWIRE PORT(S)
155
CD-ROM DRIVE
156
NETWORK INTERFACE

FIG. 1

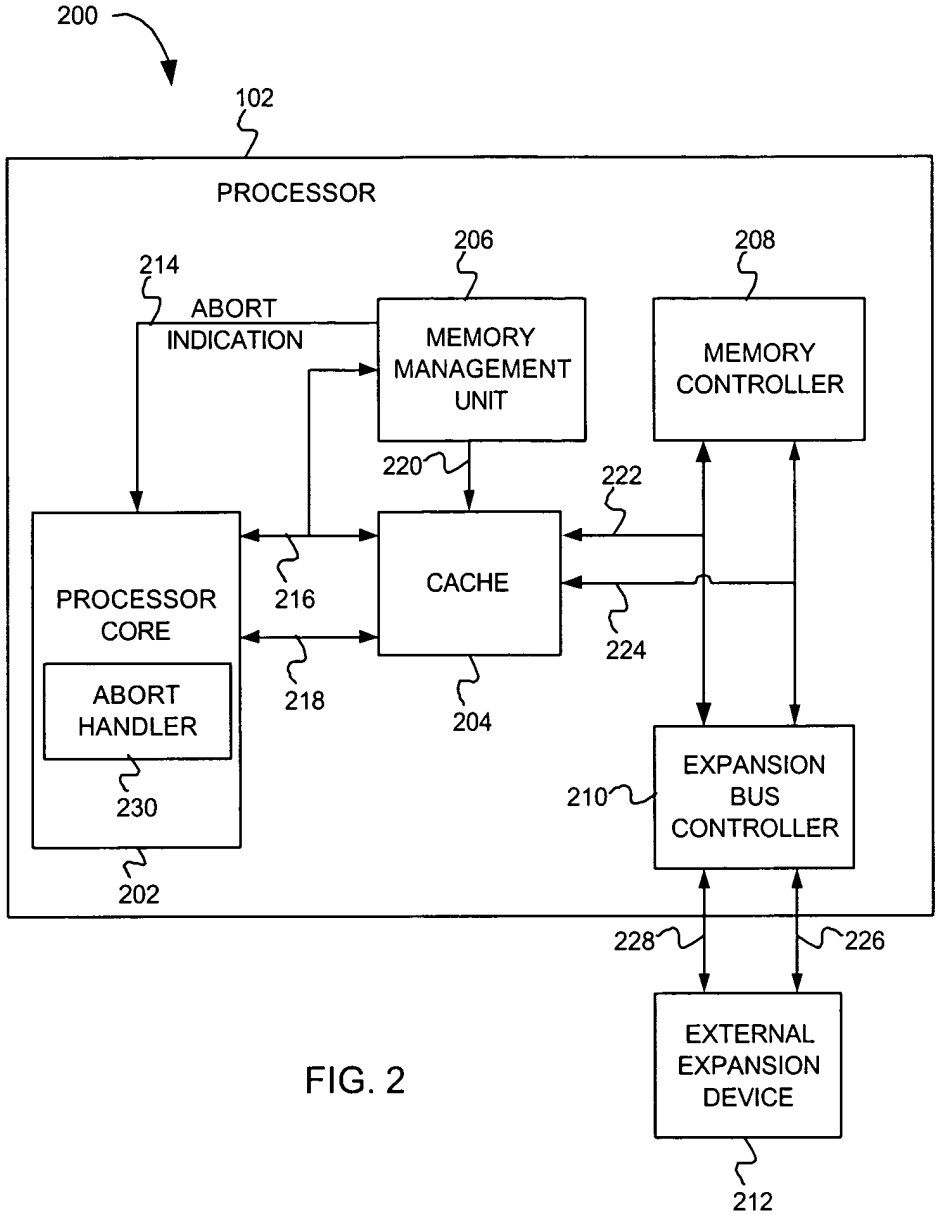
200
102
PROCESSOR
214
ABORT
INDICATION
206
MEMORY
MANAGEMENT
UNIT
208
MEMORY
CONTROLLER
220
222
PROCESSOR
CORE
ABORT
HANDLER
230
202
216
218
CACHE
204
224
210
EXPANSION
BUS
CONTROLLER
228
226
EXTERNAL
EXPANSION
DEVICE
212

FIG. 2

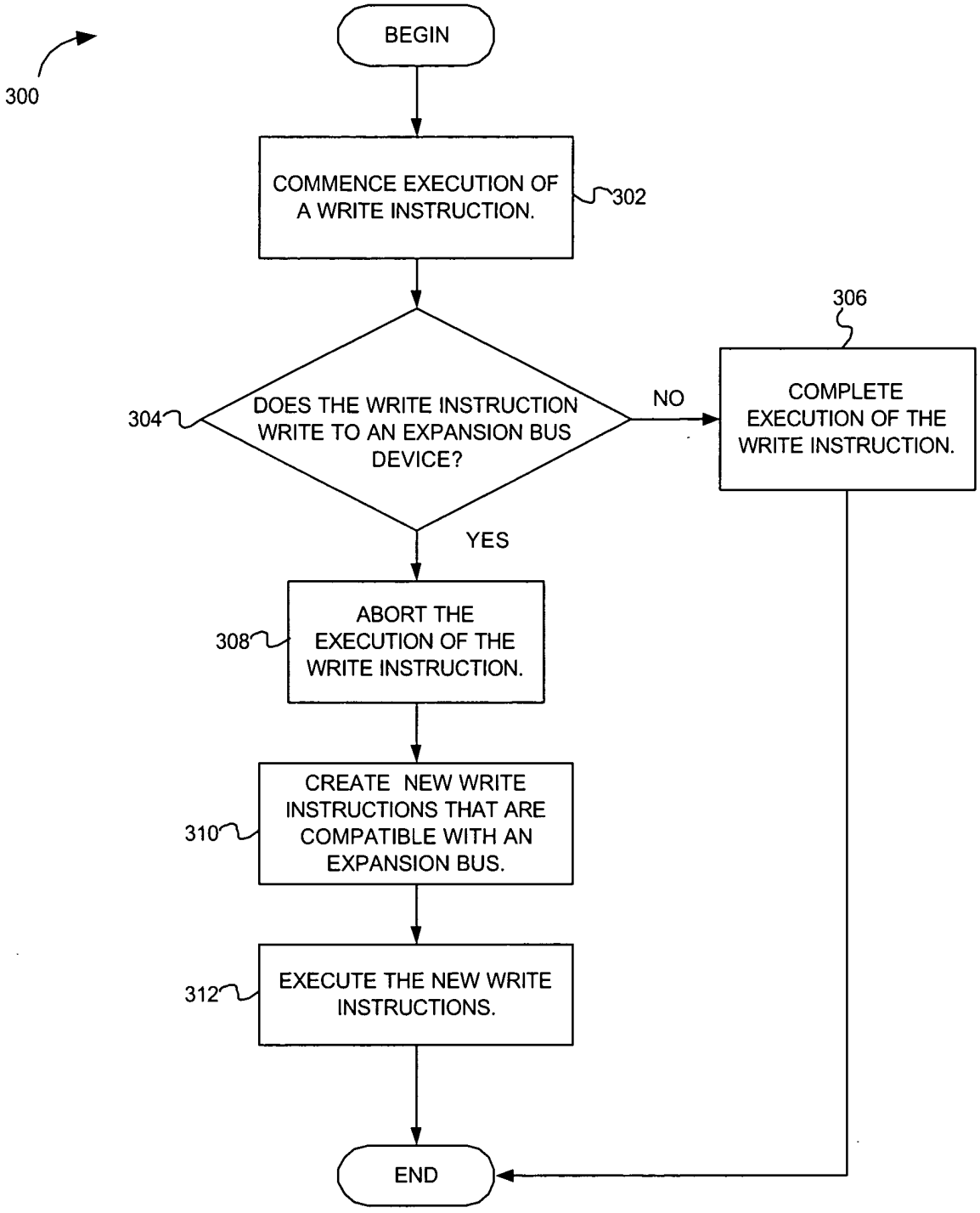
300
BEGIN
COMMENCE EXECUTION OF A WRITE INSTRUCTION.
302
DOES THE WRITE INSTRUCTION WRITE TO AN EXPANSION BUS DEVICE?
304
NO
306
COMPLETE EXECUTION OF THE WRITE INSTRUCTION.
YES
ABORT THE EXECUTION OF THE WRITE INSTRUCTION.
308
CREATE NEW WRITE INSTRUCTIONS THAT ARE COMPATIBLE WITH AN EXPANSION BUS.
310
EXECUTE THE NEW WRITE INSTRUCTIONS.
312
END

FIG. 3

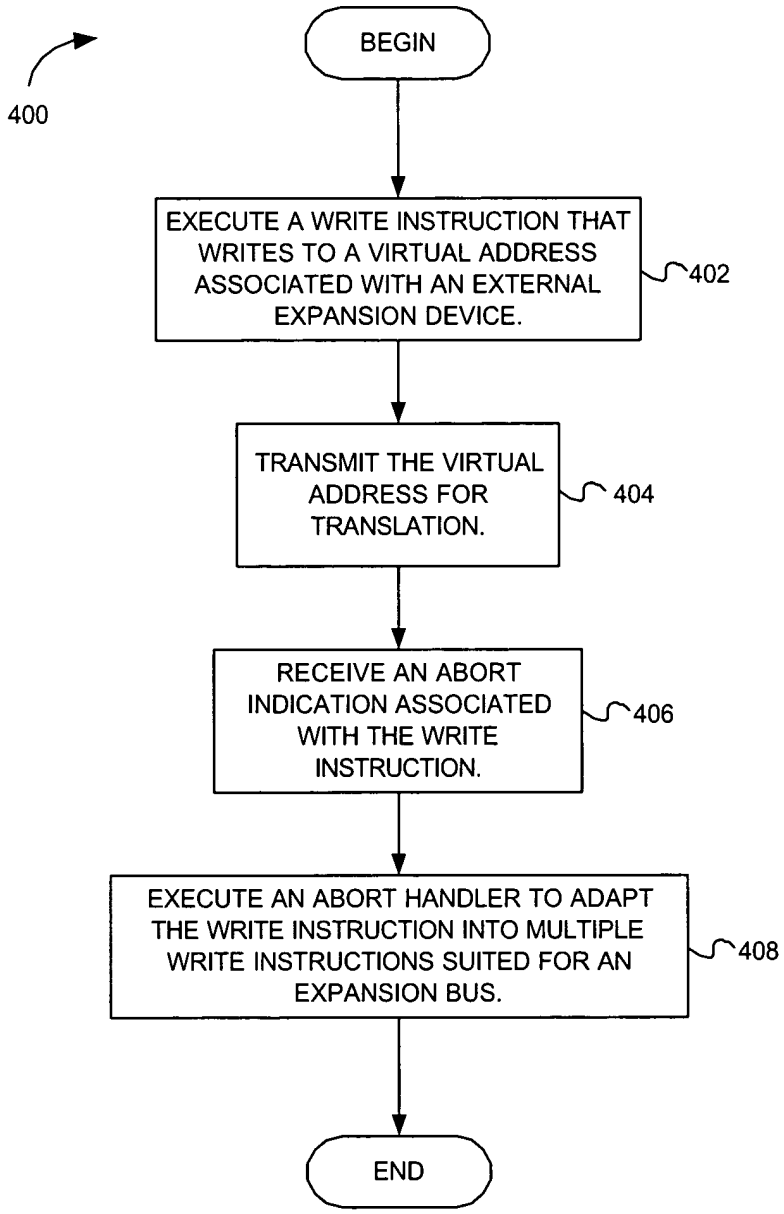
400
BEGIN
EXECUTE A WRITE INSTRUCTION THAT WRITES TO A VIRTUAL ADDRESS ASSOCIATED WITH AN EXTERNAL EXPANSION DEVICE.
402
TRANSMIT THE VIRTUAL ADDRESS FOR TRANSLATION.
404
RECEIVE AN ABORT INDICATION ASSOCIATED WITH THE WRITE INSTRUCTION.
406
EXECUTE AN ABORT HANDLER TO ADAPT THE WRITE INSTRUCTION INTO MULTIPLE WRITE INSTRUCTIONS SUITED FOR AN EXPANSION BUS.
408
END

FIG. 4

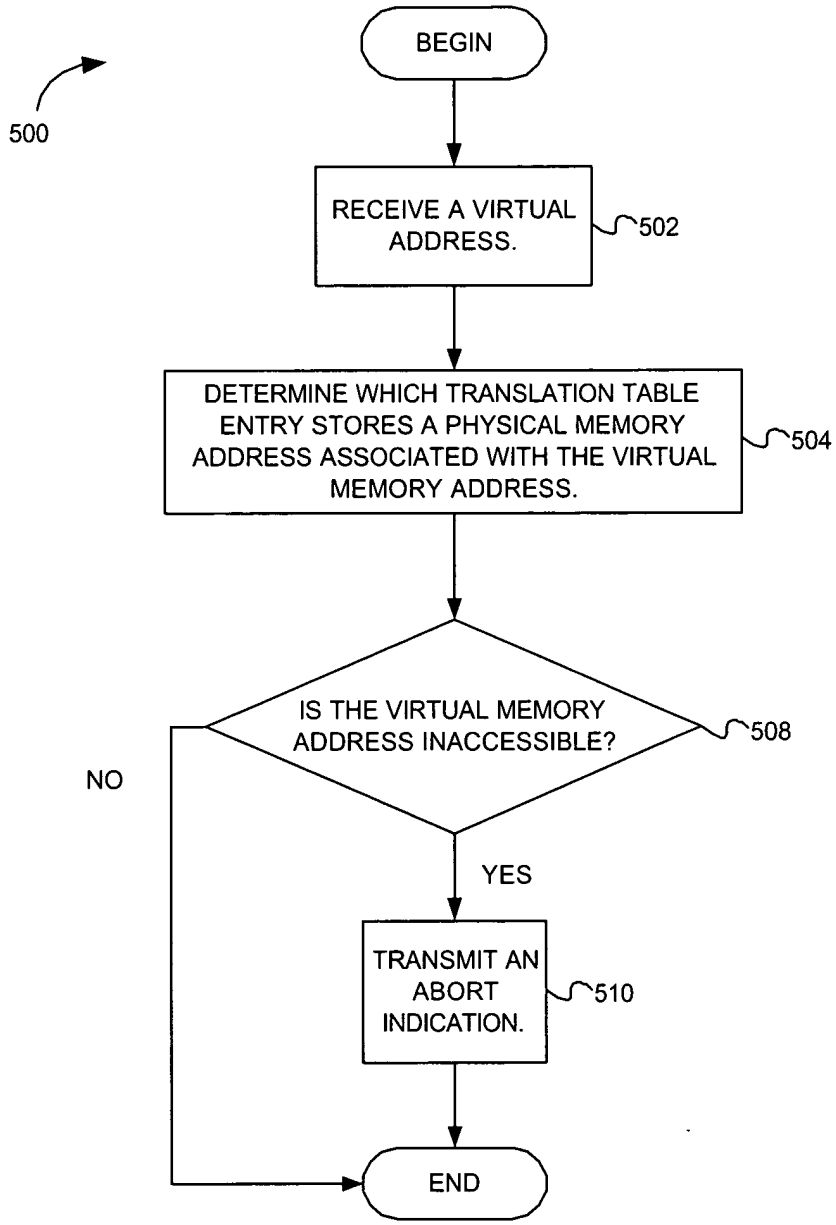
500
BEGIN
RECEIVE A VIRTUAL ADDRESS.
502
DETERMINE WHICH TRANSLATION TABLE ENTRY STORES A PHYSICAL MEMORY ADDRESS ASSOCIATED WITH THE VIRTUAL MEMORY ADDRESS.
504
IS THE VIRTUAL MEMORY ADDRESS INACCESSIBLE?
508
NO
YES
TRANSMIT AN ABORT INDICATION.
510
END

FIG. 5


```
FUNC_LABEL(expansionAbortHandler)

602 stmdb sp!,{r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,lr}

604 mrc    p15,0,r0,c6,c0,0
606 ldr    r1,[r14,#-8]
608 mov    r2,sp

610 expansionWriteEmulator

612 ldmia sp!,{r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,lr}

614 subs pc,r14,#4
```

FIG. 6

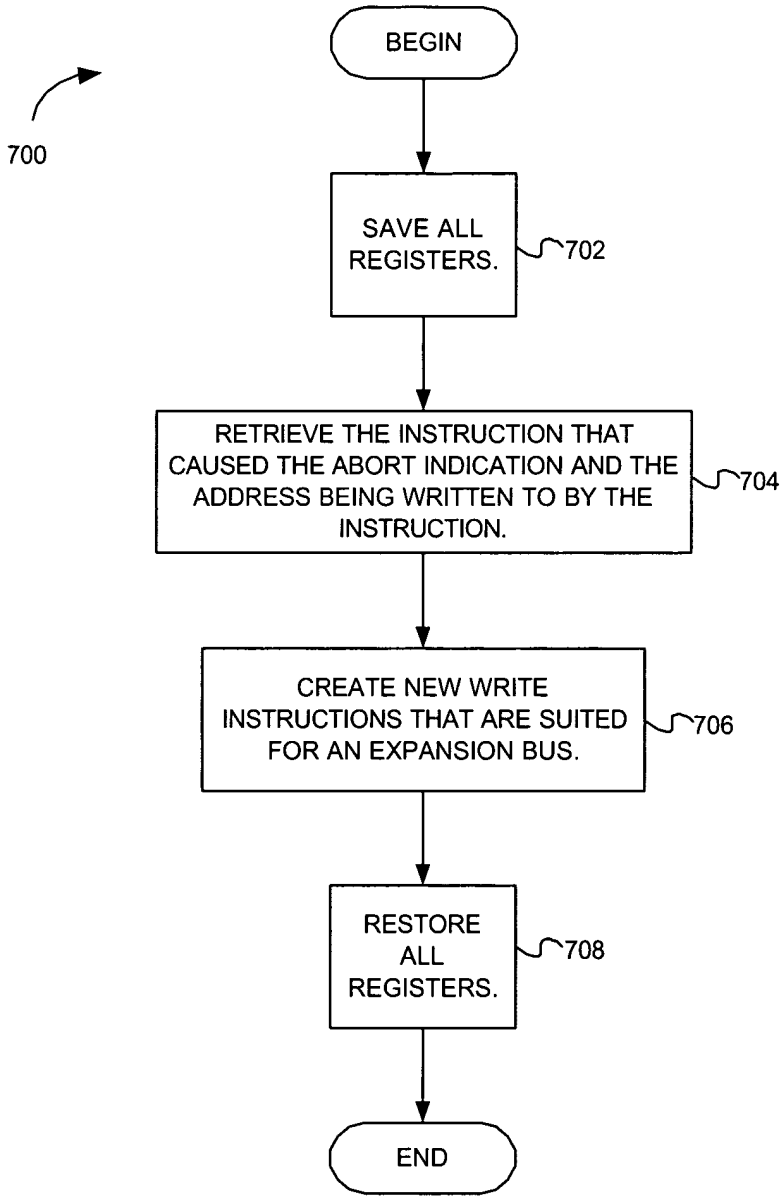
700
BEGIN
SAVE ALL REGISTERS.
702
RETRIEVE THE INSTRUCTION THAT CAUSED THE ABORT INDICATION AND THE ADDRESS BEING WRITTEN TO BY THE INSTRUCTION.
704
CREATE NEW WRITE INSTRUCTIONS THAT ARE SUITED FOR AN EXPANSION BUS.
706
RESTORE ALL REGISTERS.
708
END

FIG. 7

800

```
typedef struct
{
    unsigned long regs[NUMBER_OF_PROCESSOR_REGS] ;
} xscale_abort_regs;


void expansionWriteEmulator(unsigned long faultAddress,
                            unsigned long faultInstruction,
                            xscale_abort_regs *abortRegs)
{
   unsigned long writeVal;
802 if ( XSCALE_INSTRUCTION_IS_32BIT_WRITE(faultInstruction))
    {
806  writeVal =
          abortRegs->regs[LDR_STR_RD_BITS(faultInstruction)];
808  faultAddress &= 0x7FFFFFFF;
810  *(( unsigned short * )(faultAddress +2 )) = (writeVal & 0xFFFF);
812  *(( unsigned short * )(faultAddress )) = ((writeVal >> 16) & 0xFFFF);
    }
    return;
}
```

FIG. 8

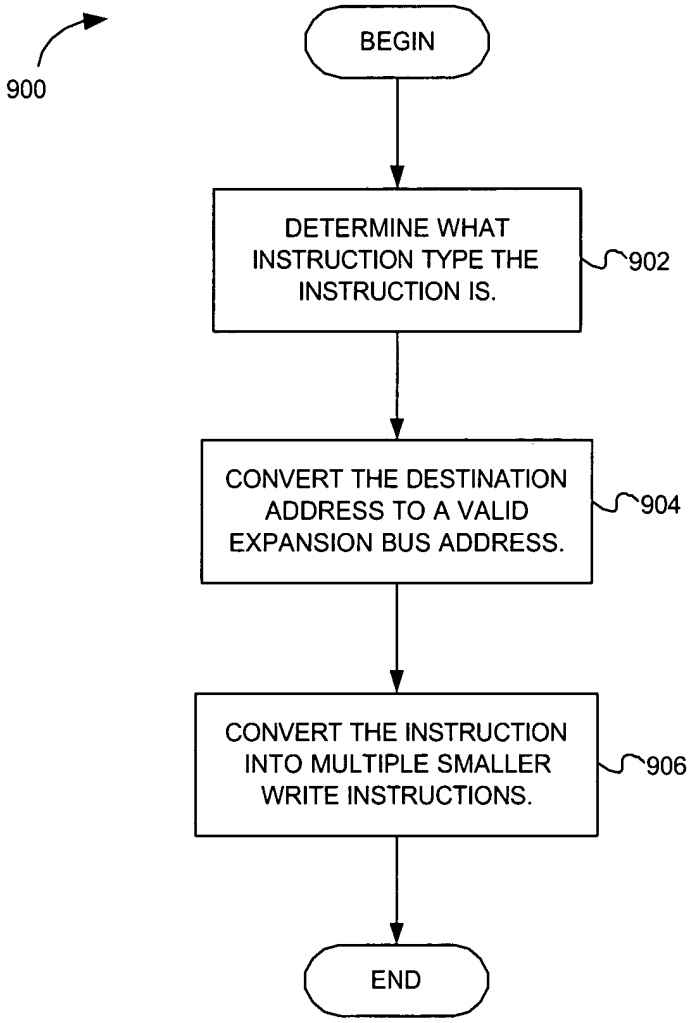
900
BEGIN
DETERMINE WHAT INSTRUCTION TYPE THE INSTRUCTION IS.
902
CONVERT THE DESTINATION ADDRESS TO A VALID EXPANSION BUS ADDRESS.
904
CONVERT THE INSTRUCTION INTO MULTIPLE SMALLER WRITE INSTRUCTIONS.
906
END

FIG. 9